

169.1736



PATENT APPLICATION

05067  
115-0  
cd

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: )  
: Examiner: NYA  
TIMOTHY JOHN LINDQUIST )  
: Group Art Unit: NYA  
Application No.: 09/587,052 )  
:  
Filed: June 2, 2000 )  
:  
For: RECONFIGURABLE VLIW )  
PROCESSOR : June 26, 2000

Assistant Commissioner for Patents  
Washington, D.C. 20231

CLAIM TO PRIORITY

Sir:

Applicant hereby claims priority under the  
International Convention and all rights to which he is  
entitled under 35 U.S.C. § 119 based upon the following  
Australian Priority Application:

PQ 0705 filed June 2, 1999

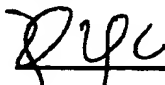
A certified copy of the priority document is  
enclosed.

Applicant's undersigned attorney may be reached in  
our New York office by telephone at (212) 218-2100. All

correspondence should continue to be directed to our address  
given below.

Respectfully submitted,

  
\_\_\_\_\_  
Attorney for Applicant

Registration No. \_\_\_\_\_  
9.

FITZPATRICK, CELLA, HARPER & SCINTO  
30 Rockefeller Plaza  
New York, New York 10112-3801  
Facsimile: (212) 218-2200  
92160



09/587,052



Patent Office  
Canberra

## CERTIFIED COPY OF PRIORITY DOCUMENT

I, ANNA MAIJA EVERETT, ACTING TEAM LEADER EXAMINATION SUPPORT & SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 0705 for a patent by CANON KABUSHIKI KAISHA filed on 02 June 1999.

I further certify that pursuant to the provisions of Section 38(1) of the Patents Act 1990 a complete specification was filed on 01 June 2000 and it is an associated application to Provisional Application No. PQ 0705 and has been allocated No. 37834/00.



WITNESS my hand this  
Eighth day of June 2000

*A. M. Everett.*

ANNA MAIJA EVERETT  
ACTING TEAM LEADER  
EXAMINATION SUPPORT & SALES

ORIGINAL

AUSTRALIA

Patents Act 1990

**PROVISIONAL SPECIFICATION FOR THE INVENTION ENTITLED:**

Reconfigurable VLIW Processor

---

Name and Address  
of Applicant: Canon Kabushiki Kaisha, incorporated in Japan, of 30-2,  
Shimomaruko 3-chome, Ohta-ku, Tokyo, 146, JAPAN

Name(s) of Inventor(s): Timothy John Lindquist

This invention is best described in the following statement:

## RECONFIGURABLE VLIW PROCESSOR

### Field of Invention

The present invention relates to a novel processor architecture, and in particular, to a reconfigurable Very Long Instruction Word ("VLIW") processor.

5       The invention has been developed primarily for use in computer graphics applications in which pipeline processing of streamed data is desirable, and is described herein with reference to this example. However, it will be appreciated that the invention is not limited to use in such applications.

### Background of Invention

10       In a basic computer system, a single, multi-function processor is used to implement all instructions provided by a computer program. The usual way to improve the speed of such a processor is to increase the speed of the clock supplied thereto. However, as is well known in the art, there are physical and material limitations upon the clock speed at which any particular processor can be driven.

15       One way around the problem is pipelining, in which each program instruction is broken down into the steps required for its execution. Such steps can include, for example, fetching an instruction from memory, decoding the instruction, fetching data required for the instruction from memory, executing the instruction using the retrieved data, and writing the result to memory. By implementing each of these steps in a  
20       separate processor module associated with the main processor, it is possible to drastically increase the throughput of instructions.

One situation in which pipelining is particularly advantageous is the processing of large amounts of streamed data. An example of this is image processing, in which

filtering operations frequently require the repeated application of a particular instruction or instructions to each pixel in an image.

There are also some disadvantages in pipeline processing. For example, the results of a conditional branch instruction will not be known until at least the execution  
5 step mentioned above. This means that it is not possible to commence fetching the next instruction until a number of clock cycles have been wasted. When overheads associated with implementing pipelining are taken into account, this architecture can result in less efficiency than a well-designed single processor running at a similar speed.

Another method of increasing the speed at which at least certain types of  
10 programs run is to implement a plurality of processor units in parallel. A particular type of parallel arrangement is the VLIW processor, a simple processor architecture in which the instruction for each functional unit is contained within a dedicated field of a VLIW. Such a processor "simple" because, although it processes a number of operations in parallel, all dependency and scheduling issues are handled by a program  
15 compiler, rather than in hardware. However, as with pipelined processors, VLIW processors can be relatively inefficient users of resources when running programs having particular operational characteristics.

In the context of graphics operations, generally the same operation is applied over and over again to each element of an input data stream. Often, the operations  
20 applied are complex and composed of a tree of primitive operations. An example of this is compositing, where several layers of objects are used to provide a final output image. For each output pixel, there is a multi-level compositing tree, made of primitive compositing operators. Likewise, calculation of colour space conversion requires a short computation tree, and convolution requires a hierarchy of multiplications and

additions for each output pixel. Unfortunately, single level VLIW processors and single width pipelined processors do not necessarily provide an optimal solution to the need for additional processing speed in this type of application.

It is an object of the present invention to provide a reconfigurable VLIW processor that overcomes or at least ameliorates one or more of the disadvantages of the prior art.

### Summary of Invention

In accordance with one aspect of the present invention there is provided a computer processor including a plurality of processing units interconnected by communication means, the communication means being configurable such that the processing units can selectively be arranged in at least first and second distinct configurations, the first distinct configuration including at least two of the processing units arranged in a pipeline at least two layers deep, and the second distinct configuration including at least two of the processing units arranged in a parallel arrangement at least two processing units wide.

Preferably, in the first distinct configuration the pipeline is at least two processing units wide. Similarly, it is preferred that the parallel arrangement is pipelined at least two layers deep in the second distinct configuration.

In a preferred form, the communication means includes a data bus, the data bus being configurable to selectively interconnect inputs and outputs of the processing units, thereby to form the first and second configurations.

Other aspects of the invention will become apparent from a reading of the following detailed description, and by referring to the numbered paragraphs at the end of this document.

### **Brief Description of Drawings**

Preferred embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Fig. 1 shows a single tree instruction for use in known IBM VLIW  
5 architecture;

Fig. 2 shows the substeps defined by the tree instruction shown in Fig. 1 broken into branches, conditional branches and primitive operations;

Fig. 3 is a schematic diagram of a plurality of processing units interconnected by a bus, according to a preferred embodiment of the invention;

10 Fig. 4 shows the architecture of Fig. 3 with the processing units arranged in a first configuration;

Fig. 5 shows the architecture of Figs. 3 and 4 with the processing units interconnected to form a second configuration distinct from the first configuration;

Fig. 6 is a schematic diagram of the processor architecture shown in Figs. 3 to  
15 5 with the processing units interconnected to form a third configuration distinction from the first and second configurations; and

Fig. 7 shows a tree diagram showing a single VLIW spanning 22 functional units, with internal branching.

### **Detailed Description of Preferred Embodiments**

#### **20 Prior Art VLIW Processors**

A conventional VLIW ("Very Long Instruction Word") processor is composed of a fixed number of functional units arranged as a single static computational layer. It is the job of an associated compiler to extract any potential parallelism from an executed program in order to feed this computational structure. Whist it may appear difficult to



extract parallelism from a sequential program, if a speculative execution approach is used, it is frequently possible to obtain a relatively large amount of instruction level parallelism. Speculative execution is a term used to describe the function of executing instructions that are not yet current, in the statistical expectation that these instructions may soon be reached by the flow of the program. A typical scenario is the forward execution of instructions that are on both possible paths emanating from a branch instruction. Instructions on both paths are executed, but the results are inhibited from being written to registers or memory until it is possible to deduce the taken path.

A specific example of the extraction of parallelism for a given program will now be described in relation to IBM's tree-based approach. In the IBM VLIW architecture, a program consists of a sequence of tree instructions. A tree instruction is a compound instruction, in the form of a tree, which maps out the flow of a program, dividing into separate paths at each branch point. Operations that lie on the segments between the branches must not have dependencies within the same tree.

In a more formal sense, each tree instruction is a collection of primitive operations that have no dependencies and can be executed in parallel; that is, if there are adequate functional units to process all the primitive operations in the tree in parallel, then the tree represents a single VLIW. The tree instruction corresponds to an unlimited multiway branch with multiple branch targets and an unlimited set of primitive operations.

Where the number of simultaneous branch instructions, or the number of primitive operations, exceeds the resources of the implementation, then the tree can be pruned to smaller subtrees that execute sequentially. Each such subtree is itself a parallel VLIW.

A single tree instruction is shown in Fig. 1. It will be appreciated that, within this tree instruction, all operations and all branches are independent and executable in parallel. Accordingly, based on an evaluation of the branch conditions, the taken path can be determined at execution time. Those operations within the VLIW that are on the taken path are allowed to complete, whilst those not on the taken path are inhibited from committing results to storage or registers.

There will necessarily be some resource wastage in this architecture, because operations are performed speculatively, and only those that actually lie on the taken path have their results committed. Also, only operations without dependencies are allowed within the same tree (that is, within the same VLIW); once a dependency is detected, then the dependent operation must be placed in a subsequent tree (VLIW).

Turning to Fig. 2, it will be seen that for the tree instructions shown in Fig. 1, 22 functional units are required to execute the complete tree instruction in parallel. However, at most 7 units (for the path L0 to A) actually have results committed. In addition, trees are decomposed where necessary so that subtrees fit the fixed length of the VLIW implementation. This requires pruning at conditional branch points so that the resultant subtree is containable within the VLIW. Where a subtree is smaller than a VLIW, the unallocated units are wasted for that cycle.

### **Preferred Embodiments of the Invention**

In the preferred embodiment, there is provided a plurality of N processing units which can dynamically be configured to any required logical network configuration. The preferred form of each processing unit is an SIMD execution unit, similar to an Intel MMX execution unit. Such a processing unit is capable of treating a data word as a set of sub-words, to which it uniformly applies the same operation. For

example, in the graphics context a 32-bit data word would comprise 4 independent 8-bit channels. The processing units would then apply the same operation to each channel concurrently.

In order to support dynamic reconfiguration of processing units, a flexible  
5 interconnecting communication network is required. In a relatively simple implementation, a single unified bus is used to interconnect inputs and outputs of the respective processing units. Such a communication arrangement has the disadvantage that it throttles total system performance, because the bandwidth of a single communication channel must be time multiplexed between connections. A better, if  
10 more complicated, network involves the use of a mesh network, although multiple toroidal meshes, a cross-point network or any other suitable data communication network can also be used.

It will be appreciated by those skilled in the art that, for minimum latency, a connection network must provide no more than  $3N$  connections at any point in time for  
15 a typical processing unit setup. This is because each of the  $N$  processing units requires two inputs and generates one output. A mesh that connects each functional unit with 4 or more neighbours should, with an appropriate routing algorithm which will be apparent to those skilled in the art, allow simultaneous communication between all active processing unit source and destination combinations with minimal hops. It will  
20 be appreciated by those skilled in the art the specific mesh arrangement (where a mesh is used) can vary from application to application. In particular, in any network, the number of neighbouring functional units to which each functional unit is connected need not be it, and indeed, can be any suitable number for a given application.

Notwithstanding its disadvantages, the unified bus embodiment is relatively easy to conceptualise in operation, and provides a simple implementation in situations where some system degradation is acceptable. A specific example is shown in Figs. 3 to 6, where  $N=16$ , and processing (or functional) units FU1-FU16 are interconnected by a unified bus 300. The bus also connects to storage units, but for simplicity the storage units and their network connections are not shown.

From such a collection of processing units, it is possible to construct any logical network ranging from a single level fixed-width 16-unit VLIW, through to a 16-level deep, 1 unit wide pipeline. In the middle of this range, where a logical network comprising several pipelined stages of VLIWs may be envisaged, it is also possible to have different width VLIWs at each level. However, it will be appreciated by those skilled in the art that, due to instruction fetch and decode issues, all levels in a pipeline structure will usually have an equal number of processing units.

As shown in Fig. 4, when a plurality of processing units FU1-FU16 are configured to form a single computational layer 400, the processor architecture simulates a standard VLIW processor. In this case, after the execution of an initial VLIW, all processing units must be loaded with new operations for implementing a subsequent VLIW. Only the results from a valid path through the VLIW are committed to memory, as discussed in more detail below under the subheading "Pipelined VLIWs - Parallel Vector Computer".

Turning to Fig. 5, a second configuration of the processing units FU1-FU16 is shown. In this case, the processing units FU1-FU16 are arranged in a four by four configuration 500, forming a pipeline four units deep and four units wide.

In the case of the pipelined VLIW configuration, the functional units of the processor are arranged in a layered network. This configuration offers relatively good potential for computational loops. Data flow 502 through the layers of the pipeline is indicated with respect to corresponding data ensembles that are being processed. As shown, a data ensemble enters a first level 504 and processing begins. The results from the first level 504 feed to a second level 506, whilst data for the second ensemble is accepted by the first level 504. This process is repeated for each subsequent layer in the pipeline until the result is output from a fourth layer 508.

The greatest advantage of a pipeline of this construction is obtained when the processing requirements of the loop are satisfied by the available resources, such that data ensembles are processed at the rate of one ensemble per clock. However, longer loops are feasible at reduced throughput by overlaying the loop code over multiple passes through the pipeline discussed in more detail below, as will be understood by those skilled in the art.

It should be noted that this arrangement requires a special implementation of storage elements (registers), which themselves must assume a quasi-pipelined nature. However, this requirement only applies to a subset of such elements, so the requirement does not lead to a substantial increase in register implementation requirements.

It should also be noted that it is possible to feed multiple data items into the first stage of the VLIW pipeline, so long as the implemented architecture allows for a number of units with load/store functionality. The architecture of the memory access interface will be the determining factor in such an implementation, as will be understood by those skilled in the art.

Turning to Fig. 6, if little or no parallelism can be extracted from a program, and particularly where each sequential operation is dependent on the previous operation, it is still possible to arrange the processing units FU1-F16 as a narrow pipeline 600 and obtain a throughput of one result per clock.

## 5 **Pipelined VLIWs - Parallel Vector Computing**

By configuring the processing units FU1-F16 in a logical network configuration of pipelined VLIWs, such as the arrangement shown in Fig. 5, a parallel vector computer is implemented. Such a configuration is desirable for the processing of computational loops which contain no branching within the loop, and which need to  
10 operate on one or more data input streams. It will be appreciated that this arrangement is not limited to short loops that fit within the available number of pipeline levels, although maximum performance is achieved for those loops that do fulfil this constraint. Longer loops are accommodated by wrapping the code over the pipeline stages as the data ensembles pass through each loop segment, in a manner which will be apparent to  
15 those skilled in the art.

It is up to the software compiler to determine the optimal width of the VLIWs. Those skilled in the art of processor design will appreciate that it is wasteful implementing a four unit wide VLIW if only enough instruction level parallelism can be found to utilize two or three processing units at each level. Provided that the length of  
20 the loop warrants such action, it is better to limit the width of the VLIW thus allowing a greater pipeline depth.

The operation of overlaying loop code segments onto the pipeline array is described in the following VLIW sequence:

	VLIW0	mov r0, [r15]:
25	VLIW1	mov r1, [r15+8]:

```
VLIW2          mov r2, [r15+16]: sub r1, r0:
VLIW3          mov r3, [r15+24]: mul r1, r8:
VLIW4          mov r4, [r15+256}: sub r3, r2: sra r1, 6
VLIW5          mov r5, [r15+264]: mul r3, r8: add r0, r1
5 VLIW6..... mov r6, [r15+272]: sub r5, r4: sra r3, 6
VLIW0          mov r7, [r15+280]: mult r5, r8: add r2, r3
VLIW1          sub r7, r6: sra r5, 6: sub r2, r0
VLIW2          mul r7, r8: mul r2, r9: add r4, r5
VLIW3          mov r12, [r13]: sra r7, 6: sra r2, 6
10 VLIW4..... mov [r10], r4: add r10, 1: add r15, r7
```

This loop code is twelve VLIWs in length, but the computational capacity of the configuration shown is only 7 VLIWs deep. The code wraps back to VLIW0 after the seventh VLIW (VLIW6). This occurs as soon as the seventh data ensemble has progressed through VLIW0. Further ensembles are blocked from entering the pipeline until the current set has completely traversed the loop. The throughput for this example is 7 ensembles every 12 clocks. The branch condition at the end of the loop is not shown.

Overlaying loop code introduces some complexities into the instruction issue logic, as will be appreciated by those skilled in the art, but this drawback is outweighed by the improved performance.

In the situation where all the loop code can be satisfied by available resources without overlay, the instructions loaded into each processing unit are static for the duration of the complete data set that passes through the pipeline. When overlaying is required, the instructions loaded in the functional units do not change whilst the current data subset is feeding into the pipeline. When the pipeline is full, the VLIWs at the

head of the pipeline are issued with new instructions as the last ensemble passes them. This is the process of loop code overlay.

There are some fundamental requirements for the implementation of pipelined VLIW code. One of the most basic conditions is that there can not be any change in program flow between VLIWs, for the obvious reason that one cannot implement a pipeline if the pipeline elements change from one cycle to another. However, it is possible to allow changes of program flow within a VLIW itself, but only if such branch paths are completely contained within the VLIW, with all final branch targets pointing at the same target VLIW (see discussion of prior art VLIW architecture above). This represents a condition of small branches from the main path which subsequently rejoin the main path. This is described graphically in Fig. 7, which shows a single VLIW, spanning 22 functional units, with internal branching. All branches and paths lead to the same destination "A", which is simply the next VLIW.

Another important condition for pipelining is that, in the course of data flow through the pipeline, storage locations (eg, registers and memory) must be written before they are read, or at least not read before they are written. In effect, this means it is acceptable to read and then write to the same storage locations as an atomic operation within a single VLIW, for operations such as XOR (r1, r1). This condition is the opposite to the case of speculative execution within a VLIW, where storage must be read before it is written, because all the operations execute in parallel.

The implication of this condition is that there should be no inter-loop dependencies. In other words, it is not desirable to precompute values for a successive loop iteration whilst in the current iteration. In the preferred embodiment where the processor will be used to process graphical operations, there is usually no need for such



feed-forward of data calculations. For instance, in compositing, colour space conversion, and convolution, the resulting output pixel is not dependent on the previous result. But for some processes, such as error diffusion, this constraint must be considered.

5           In order to support the pipelined architecture, storage registers must be aliased, such that each ensemble has a private register set, even though the registers for each ensemble are referenced through the same or at least suitably related names.

As an example, assume that there are four levels in the pipeline, so that there is a capacity of four concurrent computation ensembles. Each ensemble has a private  
10 register set, so that when an instruction specifies R7, the actual location accessed would be R7\_0 for ensemble 0, R7\_1 for ensemble 1, and so on. Although this would appear to massively increase the number of registers required, this is not the case because every register must be aliased as many times as there are pipeline levels. For any non-over-laid pipeline, the maximum number of storage locations required is equal to the  
15 number of functional units used in the pipeline multiplied by two inputs per unit multiplied by the number of pipeline levels. For a 4 level pipeline with 16 functional units in an architecture that has 128 nominal registers, only 128 (rather than  $128 \times 4 = 512$ ) actual aliased locations are required. In reality, the total required would normally be less than the maximum number, probably by a factor of one half, because it can be  
20 assumed that functional units normally produce results that subsequent pipeline layers operate on. Accordingly, by using a virtual register allocation scheme for register aliasing, storage requirements are minimised. Each private register set is retained until the associated ensemble has successfully left the pipeline, at which stage it is freed for the ensemble that is in the process of entering the pipeline.

It will be appreciated by those skilled in the art of the computer architecture that writes to memory locations will also be pipelined. Write data is only allowed to advance through the pipeline on the completion of the loop, in that all write data for a single iteration is only committed to memory when the iteration is complete. This  
5 ensures that pipelined data that lies beyond the loop bounds is not written to memory.

It will be understood that the particular number of processing units is, in the broadest form of the invention, arbitrary. For example, as few as two processing units can be used, configurable between two-by-one and one-by-two arrangements. However, it will also be appreciated that greater efficiencies are usually to be found  
10 using larger numbers of processing units.

It will also be understood that the number of available configurations may be as low as two, even where relatively large numbers of processing units (eg. > 16) are used. For example, for a particular combination of likely hardware tasks, it may be determined that 20 processor units switchable between a 20-wide VLIW and four wide  
15 by five deep pipelined VLIW will provide the best overall performance. In particular, by only allowing a relatively small number of configurations, the communications network required to switch between configurations can greatly be reduced in complexity. However, in some cases, multiple configurations may still be desirable, notwithstanding the added complexity.

20 The preferred embodiment of the invention allows a compiler to configure interconnections between processing units in such a way that processing of different data types is improved. For example, image processing tasks typically benefit from a VLIW architecture, whilst other data processing tasks work most efficiently with particular pipelining depths. By allowing dynamic switching between different VLIW

widths and pipelining depths, the present invention offers a commercially significant improvement over prior art multi-purpose processors and specialised single-purpose processors.

Various exemplary aspects of the present invention are set out in the following numbered paragraphs:

1. A computer processor including a plurality of processing units interconnected by communication means, the communication means being configurable such that the  
5 processing units can selectively be arranged in at least first and second distinct configurations, the first distinct configuration including at least two of the processing units arranged in a pipeline at least two layers deep, and the second distinct configuration including at least two of the processing units arranged in a parallel arrangement at least two processing units wide.
- 10 2. A computer processor according to paragraph 1, wherein, in the first distinct configuration, the pipeline is at least two processing units wide.
3. A processor according to paragraph 1 or 2, wherein, in the second distinct  
15 configuration, the parallel arrangement is pipelined at least two layers deep.
4. A processor according to any one of the preceding paragraphs, wherein the communication means includes a data bus configurable to selectively interconnect the processing units into at least the first and second configurations.
- 20 5. A processor according to paragraph 4, wherein each of the processing units includes control means for selecting one or a plurality of data reception and transmission modes, the first and second distinct configurations being selectable by

altering a control signal provided to each of the control means by an instruction controller associated with the computer processor.

6. A processor according to paragraph 4, wherein the data bus is a packet-based bus and each of the processing units is adapted to place data on and take data from the packet-based bus, the first and second configurations being selectable by manipulating packet addressing on the packet-based bus.

7. A processor according to any one of the preceding paragraphs, wherein program instructions are provided to the processing units in a variable length Very Long Instruction Word ("VLIW") format, the configuration of the processor being selected on the basis of the length of the VLIW.

DATED this Second Day of June, 1999

15

**Canon Kabushiki Kaisha**

Patent Attorneys for the Applicant

**SPRUSON & FERGUSON**

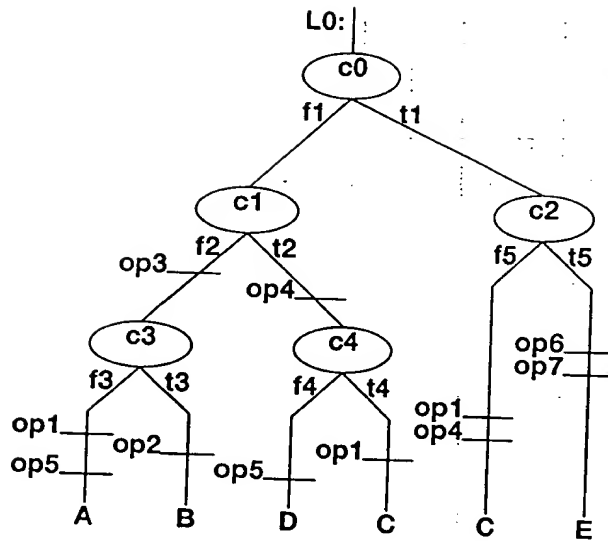


FIG 1 (PRIOR ART)

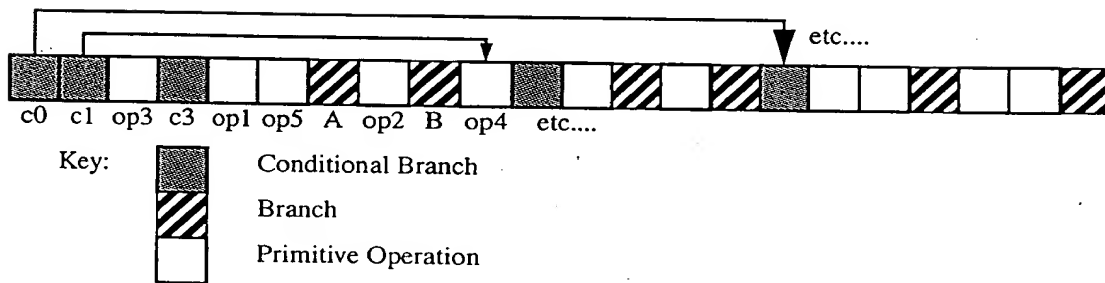


FIG 2 (PRIOR ART)

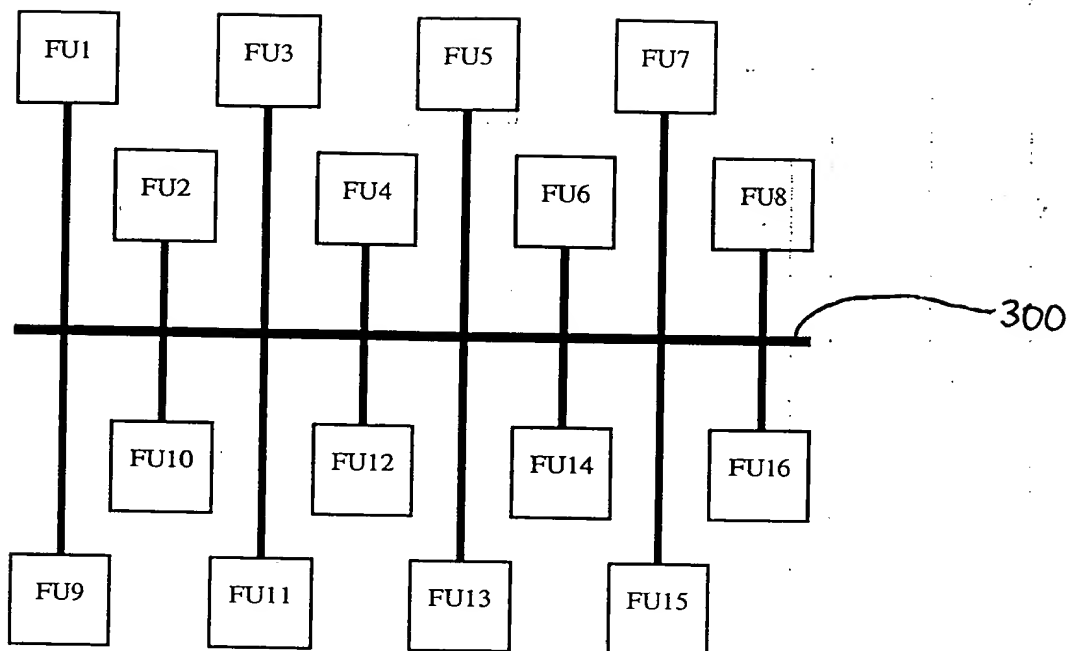


FIG 3

Instruction Word



VLIW0

Processing Units



400

FIG 4

FIG 5

Instruction Word

Processing Units

Data Ensemble

VLIW0

VLIW1

VLIW2

VLIW3

500

FU1

FU2

FU3

FU4

504

FU5

FU6

FU7

FU8

506

FU9

FU10

FU11

FU12

FU13

FU14

FU15

FU16

508

in3

in2

in1

in0

502

Instruction Word

Processing Units

Data Ensemble

IW0

IW1

IW15

600

FU1

FU2

...

FU16

in15

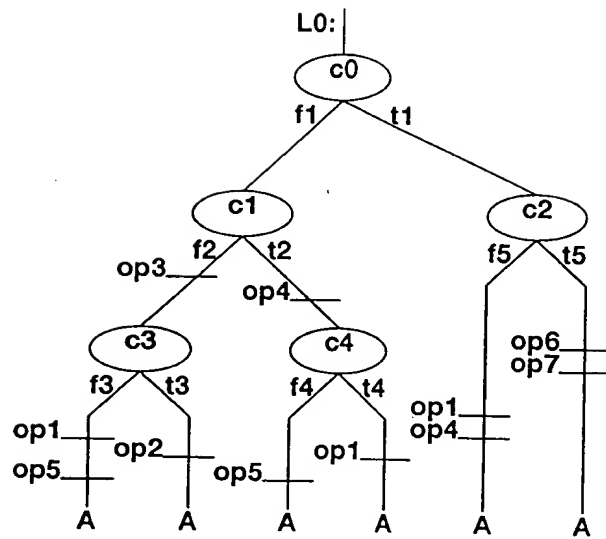
in14

in....

in0

FIG 6



FIG 7